DAM
Desarrollo de Aplicaciones Multiplataforma
2º Curso

AD
Acceso a Datos

UD 2
Manejo de Ficheros
Parte 4 - JSON

IES BALMIS
Dpto Informática
Curso 2019-2020
Versión 1 (03/2019)

UD2 – Manejo de ficheros

ÍNDICE

13. Ficheros JSON

Un flujo de datos en formato **JSON** se representa mediante objetos JSON, los cuales constan de una o más parejas "nombre":valor encerradas entre llaves {}. Las parejas se separan entre sí mediante comas.

El valor puede ser:

- **Simple**: Boolean, Number or String
- **Compuesto**:
  - JSONObject : Un objeto JSON con sus correspondientes {}
  - JSONArray : Un Array de valores simples o compuestos delimitados con []

JSONObject (SINTAXIS)	{ Object }
JSONArray (SINTAXIS)	[{ Object }, { Object }, { Object }]
JSONObject con valor Boolean	{ "nombre" : true }
JSONObject con valor Number	{ "nombre" : 2 }
JSONObject con valor String	{ "nombre" : "valor cadena" }
JSONObject con valor Array	{ "nombre" : [2, 7, 15] }
JSONObject VARIOS VALORES	{ "nombre1" : 2, "nombre2" : "valor cadena"}

Ejemplo de JSONObject con valores simples:

```
{
    "nombre": "Juan",
    "edad": 30,
    "encargado": false
}
```

Ejemplo de JSONObject que contiene el valor de "empleados" que es compuesto de un JSONArray de varios JSONObjects:

```
{
    "empresa": "Computer Intelligence",
    "empleados":[
        {"nombre":"John", "apellido":"Doe"},
        {"nombre":"Anna", "apellido":"Smith"},
        {"nombre":"Peter", "apellido":"Jones"}
    ]
}
```

Para ver el árbol de forma visual que genera el JSON podemos utilizar

JSON Viewer Online
http://jsonviewer.stack.hu/

13.1.- Tipos de API para procesar flujos JSON

Existen varios:

JSR (Java Specification Request) API	import **javax.json**
Gson API	import **com.google.gson.Gson**
Jackson API	import **com.fasterxml.jackson.core**

Nosotros trabajaremos principalmente con JSR API (javax.json).

Java API for JSON Processing
https://javaee.github.io/jsonp/
Java API – javax.json - Reference
https://javadoc.io/doc/javax.json/javax.json-api/1.1.4

Con **javax.json**, para leer un archivo JSON utilizaremos:

Json

Método
static JsonReader **createReader**(new StringReader(String que contiene el json))

JsonReader

Método
JsonObject **readObject**() //devuelve todo el flujo en un único objeto json

JsonObject

Método
String **getString**("name") // extrae el value (String) de la pareja cuyo name es "name"
Boolean **getBoolean**("name") // extrae el value (Boolean) de la pareja cuyo name es "name"
Integer **getInt**("name") // extrae el value (Integer) de la pareja cuyo name es "name"
JsonObject **getJsonObject**("name") // extrae el value (JsonObject) de la pareja cuyo name es "name"
JsonArray **getJsonArray**("name") // extrae el value (JsonArray) de la pareja cuyo name es "name"

Principalmente usaremos los siguientes objetos y métodos:

Json

Método	Descripción
createArrayBuilder	Método estático que devuelve un JsonArrayBuilder
createObjectBuilder	Método estático que devuelve un JsonObjectBuilder
createWriter(OutputStream)	Devuelve un JsonWriter (si no queremos salida “pretty”)
createWriterFactory(Properties)	Devuelve un JsonWriterFactory (si queremos salida “pretty”)

JsonObjectBuilder

Método	Descripción
add(“name”, value)	Añade una pareja al jsonObjectBuilder (devuelve otro JsonObjectBuilder por lo que se puede usar en cascada)
build()	Devuelve un JsonObject, que luego podemos escribir

JsonArrayBuilder

Método	Descripción
add(JsonObjectBuilder)	Añade un JsonObjectBuilder al JsonArrayBuilder
build():	Devuelve un JsonArray, que luego podemos escribir

JsonWriterFactory (Por si queremos que salga la salida "pretty")

Método	Descripción
createWriter(OutputStream)	Devuelve un JsonWriter

JsonWriter

Método	Descripción
write(JsonStructure value)	Sirve para escribir tanto JsonArray como JsonObject
writeArray(JsonArray array)	
writeObject(JsonObject object)	

La Jerarquía de objetos de tipo Json es:

```
JsonValue (El más general)
      → JsonNumber
      → JsonString
      → JsonStructure
            → JsonArray
            → JsonObject
```

13.2.- Creación de un JSON con Strings

Para crear un archivo JSON podemos usar métodos que nos permiten escribir la apertura y cierre de un JsonObject o un JsonArray, así como su contenido mediante un String. Veamos un ejemplo:

```
public class UD2Json01 {

  public static void main(String[] args) {
    StringWriter sw = new StringWriter();

    JsonGeneratorFactory jsonGeneratorFactory =
        Json.createGeneratorFactory(Collections.singletonMap(
            JsonGenerator.PRETTY_PRINTING, true));
    JsonGenerator generator = jsonGeneratorFactory.createGenerator(sw);

    generator.writeStartObject()
        .write("title","JSON-Processing With Java EE")
        .writeStartArray("chapters")
          .write("Introduction")
          .write("1. JSON and Java")
          .write("2. JSON-Processing API features")
          .write("3. The Java EE JSON Object model")
          .write("4. The Java EE JSON Streaming model")
          .write("Conclusion")
        .writeEnd()
        .write("released",JsonValue.TRUE)
        .write("length",90)
        .writeStartObject("sourceCode")
          .write("repositoryName", "JSON-Processing-with-Java-EE")
          .write("url","github.com/readlearncode")
        .writeEnd()
        .writeStartArray("complementaryCourse")
          .writeStartObject()
            .write("title", "RESTful Service with JAX-RS 2.0")
            .write("length", 120)
          .writeEnd()
          .writeStartObject()
            .write("title", "Java Enterprise Edition Introduction")
            .write("length", 130)
          .writeEnd()
        .writeEnd()
        .write("notes", JsonValue.NULL)
      .writeEnd();

    generator.close();
    System.out.println(sw.toString());
}
```

El archivo creado será:

```
{
    "title": "JSON-Processing With Java EE",
    "chapters": [
        "Introduction",
        "1. JSON and Java",
        "2. JSON-Processing API features",
        "3. The Java EE JSON Object model",
        "4. The Java EE JSON Streaming model",
        "Conclusion"
    ],
    "released": true,
    "length": 90,
    "sourceCode": {
        "repositoryName": "JSON-Processing-with-Java-EE",
        "url": "github.com/readlearncode"
    },
    "complementaryCourse": [
        {
            "title": "RESTful Service with JAX-RS 2.0",
            "length": 120
        },
        {
            "title": "Java Enterprise Edition Introduction",
            "length": 130
        }
    ],
    "notes": null
}
```

13.2.- Librería JAR

Para poder compilar el proyecto, deberemos añadirle la librería
"**javax.json-VERSION.jar**"

javax.json - Descargar
https://jar-download.com/ • Buscar "**javax.json**" • y pulsar en el botón "**Download javax.json**"

13.3.- Creación de un JSON con Objetos

Vamos a crear una lista de objetos en Java:

```
public class Objeto {
  int idObjeto;
  String nomObjeto;

  public Objeto(int idObjeto, String nomObjeto) {
        this.idObjeto = idObjeto;
        this.nomObjeto = nomObjeto;
  }

  public int getIdObjeto() {
        return idObjeto;
  }
  public void setIdObjeto(int idObjeto) {
        this.idObjeto = idObjeto;
  }

  public String getNomObjeto() {
        return nomObjeto;
  }
  public void setNomObjeto(String nomObjeto) {
        this.nomObjeto = nomObjeto;
  }

  @Override
  public String toString() {
      return "Objeto{" + "idObjeto=" + idObjeto +
                  ", nomObjeto=" + nomObjeto + '}';
  }

}
```

```
public class UD2Json02 {
  public static void main(String[] args)  {

    List<Objeto> ListaObjetos = new ArrayList<>();

    Objeto o1 = new Objeto(1,"Windows");
    Objeto o2 = new Objeto(2,"Linux");

    ListaObjetos.add(o1);
    ListaObjetos.add(o2);

    System.out.println(ListaObjetos.toString());

  }
}
```

Ahora deseamos obtener un JSONArray como:

```
[
  {"Id":1,"Nombre":"Windows"},
  {"Id":2,"Nombre":"Linux"}
]
```

Para ello, en un primer paso crearemos el JSONArray y luego recorreremos la lista añadiendo los JSONObjects, uno por cada objeto de la lista:

```
...

  JsonArrayBuilder jsonArrayB = Json.createArrayBuilder();
  for (Objeto obj : ListaObjetos) {
    JsonObjectBuilder jsonOB = Json.createObjectBuilder();
    jsonOB.add("Id", obj.idObjeto);
    jsonOB.add("Nombre", obj.nomObjeto);
    jsonArrayB.add(jsonOB);
  }
  JsonArray arrayJ = jsonArrayB.build();

...
```

Ahora para almacenarlo, usaremos createWriterFactory:

```
...

  JsonWriterFactory jsonFactory = Json.createWriterFactory(
      Collections.singletonMap(JsonGenerator.PRETTY_PRINTING,true)
    );
  JsonWriter jsonW =
      jsonFactory.createWriter(
          new FileOutputStream("./datos/salida.json")
        );
  jsonW.writeArray(arrayJ);
  jsonW.close();

...
```

13.4.- Lectura de un JSON

Un JsonArray lo podemos recorrer con un

```
for (JsonValue elemento : jsonArray) {
```

y luego podemos "castear" cada elemento a lo que queramos.
En el caso de que los elementos del Array sean objetos:

```
jsonObject = (JsonObject) elemento;
```

Cuando ya tenemos un JsonObject, si solo tiene claves y valores, podemos recorrerlo con foreach:

```
jsonObject.forEach((key, value) -> {
   System.out.println(key+":"+value);
});
```

También se puede obtener directamente el valor de un key sabiendo su tipo:

```
jsonObject.getString("nombre");
jsonObject.getInt("edad");
jsonObject.getBoolean("trabaja");
```

Analizando el origen de los datos

Lo primero que debemos saber del origen en formato JSON a leer es su tipo:

- Si comienza por `{` → `Es un JsonObject`
- Si comienza por `[` → `Es un JsonArray`

Lo segundo su origen, es decir, dónde se encuentra el contenido del JSON como por ejemplo:

- String
- File
- URL

De esta forma podremos crear funciones como:

Función	Descripción
getObjectFromString(String)	Obtiene un JsonObject a partir de un String
getObjectFromFile(String)	Obtiene un JsonObject a partir del nombre de un fichero
getObjectFromURL(String)	Obtiene un JsonObject a partir de la URL

Veamos cómo podría ser el código de cada una de ellas:

```
public static JsonObject getObjectFromString(String jsonObjectStr) {

  JsonReader jsonReader = Json.createReader(new StringReader(jsonObjectStr));
  JsonObject object = jsonReader.readObject();
  jsonReader.close();

  return object;
}
```

```
public static JsonObject getObjectFromFile(String strFile)
      throws FileNotFoundException  {
  File fichero = new File(strFile);  // Declarar fichero
  Reader objReader = new FileReader(fichero); //Crear el flujo de entrada

  JsonObject jsonObj;
  try (JsonReader jsonReader = Json.createReader(objReader)) {
    jsonObj = jsonReader.readObject();
        // Leemos el fichero completo en formato Object
  }

  return jsonObj;
}
```

```
public static JsonObject getObjectFromURL(String strConnection)
      throws MalformedURLException, IOException {

  // Conexión sin proxy
  URL url = new URL(strConnection);
  URLConnection uc = url.openConnection();
  HttpURLConnection conn = (HttpURLConnection) uc;

  // La conexión se va a realizar para poder enviar y recibir
  // información en formato JSON
  conn.setDoInput(true);
  conn.setDoOutput(true);
  conn.setRequestProperty("Content-type", "application/json");

  // Se va a realizar una petición con el método GET
  conn.setRequestMethod("GET");

  // Ejecutar la conexión y obtener la respuesta
  Reader objReader = new InputStreamReader(conn.getInputStream());
  JsonObject jsonObj;
  try (JsonReader jsonReader = Json.createReader(objReader)) {
    jsonObj = jsonReader.readObject();
      // Leemos el fichero completo en formato Object
  }

  return jsonObj;
}
```

Con proxy sería:

```
// Conexión con proxy
Proxy proxy = new Proxy( Proxy.Type.HTTP,
                         new InetSocketAddress("192.168.0.100", 8080));
URL url = new URL(strConnection);
URLConnection uc = url.openConnection(proxy);
HttpURLConnection conn = (HttpURLConnection) uc;
```

De igual manera podríamos crear funciones para leer **JsonArray**, cambiando **JsonObject** por **JsonArray** y **readObject** por **readArray**.

Función	Descripción
getArrayFromString(String)	Obtiene un JsonArray a partir de un String
getArrayFromFile(String)	Obtiene un JsonArray a partir del nombre de un fichero
getArrayFromURL(String)	Obtiene un JsonArray a partir de la URL

Aprovechando las funciones anteriormente creadas, podemos leer y recorrer un Json cuyo origen es un String:

```
{
  "nombre" : "Juan",
  "edad" : 30,
  "encargado": false
}
```

El Json anterior contiene datos JsonObject con varios pares de tipo (key, value) de una persona.

Realizaremos una aplicación que:

1) Crear String con el contenido del Json
   - Mostrar por pantalla
2) Obtener un JsonObject a partir del String
   - Mostrar contenido usando métodos getString(), getInt(), getBoolean()
3) Recorrer el JsonObject y mostrar todo su contenido
   - Mostrar el key, el value y el tipo con el método getValueType()

UD2Json03fromString

```
public class UD2Json03fromString {

    public static void main(String[] args) {
        // Crear String con el contenido del Json
        String cadenaJson = "{"+
                "\"nombre\": \"Juan\","+
                "\"edad\": 30,"+
                "\"encargado\": false"+
                "}";
        System.out.println(cadenaJson);
        System.out.println();

        // Obtener un JsonObject a partir del String
        JsonObject jsonPersona = getObjectFromString(cadenaJson);

        // Mostrar contenido de cada campo
        System.out.println("Nombre: "+jsonPersona.getString("nombre"));
        System.out.println("Edad: "+jsonPersona.getInt("edad"));
        System.out.println("Encargado: "+jsonPersona.getBoolean("encargado"));
        System.out.println();

        // Recorrer el JsonObject
        jsonPersona.forEach((key, value) -> {
            System.out.println("Clave: "+key);
            System.out.println("Tipo: "+value.getValueType());
            System.out.println("Valor: "+value);
            System.out.println();
        });



    }

    // Función para leer un JsonObject desde un String
    public static JsonObject getObjectFromString(String jsonObjectStr) {
      JsonReader jsonReader = Json.createReader(
                          new StringReader(jsonObjectStr));
      JsonObject object = jsonReader.readObject();
      jsonReader.close();

      return object;
    }

}
```

Si tenemos un JsonArray de varios JsonObject, podemos acceder a un elemento directamente a través de sus métodos:

```
jsonArray.get(1).asJsonObject().getString("Nombre"));
```

Serializar objetos

Las librerías de javax.json no disponen de un método automático para serializar objetos.

Si deseamos serializar /desserializar podremos utilizar las librerías de Gson.

Por ejemplo, si disponemos de la clase **Users**:

Users

```
public class Users {
    public int UserId;
    public String UserName;

    // Constructor
    public Users(int UserId, String UserName) {
        this.UserId = UserId;
        this.UserName = UserName;
    }

    // getters y setters
    public int getUserId() {
        return UserId;
    }

    public void setUserId(int UserId) {
        this.UserId = UserId;
    }

    public String getUserName() {
        return UserName;
    }

    public void setUserName(String UserName) {
        this.UserName = UserName;
    }

    // Sobrecarga del método toString para verlo como un JSON
    @Override
    public String toString() {
        String strObjeto;

        strObjeto = "{\"Userid\":" + UserId +
                            ",\"UserName\":\"" + UserName+ "\"}";
        return strObjeto;
    }
}
```

Ahora realizaremos una aplicación que:

1) Crear un objeto de la clase Users con los valores (1, "Juan")
2) Utilizando el método **toJson()** de la librería **Gson**, serializar los datos del objeto anterior a un String y mostrarlo por pantalla
3) A partir de un String con el contenido {'UserId':2, 'UserName':'Sergio'} obtener un objeto de la clase Users y mostrarlo utilizando su método toString()
4) Crear un ArraList de objetos Users y añadir dos objetos (3,"Ana") y (4,"María"). A continuación, serializa el ArrayList en un String con toJson y muéstralo por pantalla.

UD2Json04SerializarGson

```
import com.google.gson.Gson;
import java.util.ArrayList;


public class UD2Json04SerializarGson {

    public static void main(String[] args) {
        // Crear objeto de la clase Users
        Users userData = new Users(1,"Juan");
        Gson gson = new Gson();

        // Serializar objeto (Fichero a String)
        String userDataString = gson.toJson(userData);
        System.out.println(userDataString);

        // Deserializar objeto (String a Objeto)
        String userJson = "{'UserId':2, 'UserName':'Sergio'}";
        Users userObject = gson.fromJson(userJson, Users.class);
        System.out.println(userObject.toString());

        // Serializar una lista de objetos utilizando la combinación de:
            // gson.toJson que usa a su vez User.toString()
        ArrayList<Users> lista = new ArrayList<>();
        lista.add(new Users(3,"Ana"));
        lista.add(new Users(4,"María"));

        String userListaString = gson.toJson(lista);
        System.out.println(userListaString);

    }

}
```